Preliminaries

Datetime

In [ ]:
```
from datetime import datetime as dt
now = dt.now()
print("Analysis on", now.strftime('%Y-%m-%d'), "at", now.strftime('%H:%M %p'))
```

```
Analysis on 2023-08-19 at 00:03 AM
```

Establish CWD

Identifying the current working directory from which the data is stored:

In [ ]:
```
import os
os.getcwd()
```

Out[ ]:
```
'/Users/chasecarlson/Documents/GSCM Course Materials/GSCM 575 Machine Learning in Business/Python Pj
ojects/GSCM-575-ML/code'
```

Import libraries

Import the following core libraries to support the analysis:

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
# Used to ignore warning messages about future deprecations and improve readability
warnings.filterwarnings('ignore')
```

About the Data

About Dataset

This dataset created from a higher education institution (acquired from several disjoint databases) related to students enrolled in different undergraduate degrees, such as agronomy, design, education, nursing, journalism, management, social service, and technologies. The dataset includes information known at the time of student enrollment (academic path, demographics, and social-economic factors) and the students' academic performance at the end of the first and second semesters. The data is used to build classification models to predict students' dropout and academic success. The problem is formulated as a three-category classification task, in which there is a strong imbalance towards one of the classes. A complete data dictionary is provided in the appendix.

Data Source: https://www.kaggle.com/datasets/naveenkumar20bps1137/predict-students-dropout-and-academic-success?select=dataset.csv

License: CC0: Public Domain

Import Dataset

In [ ]:
```
df = pd.read_csv('data/student_success.csv')
df.head().transpose()
```

Out[ ]:

	0	1	2	3	4
Marital status	1	1	1	1	2
Application mode	8	6	1	8	12
Application order	5	1	5	2	1
Course	2	11	5	15	3
Daytime/evening attendance	1	1	1	1	0
Previous qualification	1	1	1	1	1
Nacionality	1	1	1	1	1
Mother's qualification	13	1	22	23	22
Father's qualification	10	3	27	27	28
Mother's occupation	6	4	10	6	10
Father's occupation	10	4	10	4	10
Displaced	1	1	1	1	0
Educational special needs	0	0	0	0	0
Debtor	0	0	0	0	0
Tuition fees up to date	1	0	0	1	1
Gender	1	1	1	0	0
Scholarship holder	0	0	0	0	0
Age at enrollment	20	19	19	20	45
International	0	0	0	0	0
Curricular units 1st sem (credited)	0	0	0	0	0
Curricular units 1st sem (enrolled)	0	6	6	6	6
Curricular units 1st sem (evaluations)	0	6	0	8	9
Curricular units 1st sem (approved)	0	6	0	6	5
Curricular units 1st sem (grade)	0.0	14.0	0.0	13.428571	12.333333
Curricular units 1st sem (without evaluations)	0	0	0	0	0
Curricular units 2nd sem (credited)	0	0	0	0	0
Curricular units 2nd sem (enrolled)	0	6	6	6	6
Curricular units 2nd sem (evaluations)	0	6	0	10	6
Curricular units 2nd sem (approved)	0	6	0	5	6
Curricular units 2nd sem (grade)	0.0	13.666667	0.0	12.4	13.0
Curricular units 2nd sem (without evaluations)	0	0	0	0	0
Unemployment rate	10.8	13.9	10.8	9.4	13.9
Inflation rate	1.4	-0.3	1.4	-0.8	-0.3
GDP	1.74	0.79	1.74	-3.12	0.79
Target	Dropout	Graduate	Dropout	Graduate	Graduate

View the dimensions of the data:

In [ ]:
```
df.shape
```

Out[ ]:
```
(4424, 35)
```

The data frame has 4,424 rows and 35 columns.

Data Preprocessing

Renaming column from 'Nacionality' to 'Nationality':

In [ ]:
```
df.rename(columns={'Nacionality': 'Nationality'}, inplace=True)
```

Data Description

In [ ]:
```
df.describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Marital status	4424.0	1.178571	0.605747	1.00	1.00	1.000000	1.000000	6.000000
Application mode	4424.0	6.886980	5.298964	1.00	1.00	8.000000	12.000000	18.000000
Application order	4424.0	1.727848	1.313793	0.00	1.00	1.000000	2.000000	9.000000
Course	4424.0	9.899186	4.331792	1.00	6.00	10.000000	13.000000	17.000000
Daytime/evening attendance	4424.0	0.890823	0.311897	0.00	1.00	1.000000	1.000000	1.000000
Previous qualification	4424.0	2.531420	3.963707	1.00	1.00	1.000000	1.000000	17.000000
Nationality	4424.0	1.254521	1.748447	1.00	1.00	1.000000	1.000000	21.000000
Mother's qualification	4424.0	12.322107	9.026251	1.00	2.00	13.000000	22.000000	29.000000
Father's qualification	4424.0	16.455244	11.044800	1.00	3.00	14.000000	27.000000	34.000000
Mother's occupation	4424.0	7.317812	3.997828	1.00	5.00	6.000000	10.000000	32.000000
Father's occupation	4424.0	7.819168	4.856692	1.00	5.00	8.000000	10.000000	46.000000
Displaced	4424.0	0.548373	0.497711	0.00	0.00	1.000000	1.000000	1.000000
Educational special needs	4424.0	0.011528	0.106760	0.00	0.00	0.000000	0.000000	1.000000
Debtor	4424.0	0.113698	0.317480	0.00	0.00	0.000000	0.000000	1.000000
Tuition fees up to date	4424.0	0.880651	0.324235	0.00	1.00	1.000000	1.000000	1.000000
Gender	4424.0	0.351718	0.477560	0.00	0.00	0.000000	1.000000	1.000000
Scholarship holder	4424.0	0.248418	0.432144	0.00	0.00	0.000000	0.000000	1.000000
Age at enrollment	4424.0	23.265145	7.587816	17.00	19.00	20.000000	25.000000	70.000000
International	4424.0	0.024864	0.155729	0.00	0.00	0.000000	0.000000	1.000000
Curricular units 1st sem (credited)	4424.0	0.709991	2.360507	0.00	0.00	0.000000	0.000000	20.000000
Curricular units 1st sem (enrolled)	4424.0	6.270570	2.480178	0.00	5.00	6.000000	7.000000	26.000000
Curricular units 1st sem (evaluations)	4424.0	8.299051	4.179106	0.00	6.00	8.000000	10.000000	45.000000
Curricular units 1st sem (approved)	4424.0	4.706600	3.094238	0.00	3.00	5.000000	6.000000	26.000000
Curricular units 1st sem (grade)	4424.0	10.640822	4.843663	0.00	11.00	12.285714	13.400000	18.875000
Curricular units 1st sem (without evaluations)	4424.0	0.137658	0.690880	0.00	0.00	0.000000	0.000000	12.000000
Curricular units 2nd sem (credited)	4424.0	0.541817	1.918546	0.00	0.00	0.000000	0.000000	19.000000
Curricular units 2nd sem (enrolled)	4424.0	6.232143	2.195951	0.00	5.00	6.000000	7.000000	23.000000
Curricular units 2nd sem (evaluations)	4424.0	8.063291	3.947951	0.00	6.00	8.000000	10.000000	33.000000
Curricular units 2nd sem (approved)	4424.0	4.435805	3.014764	0.00	2.00	5.000000	6.000000	20.000000
Curricular units 2nd sem (grade)	4424.0	10.230206	5.210808	0.00	10.75	12.200000	13.333333	18.571429
Curricular units 2nd sem (without evaluations)	4424.0	0.150316	0.753774	0.00	0.00	0.000000	0.000000	12.000000
Unemployment rate	4424.0	11.566139	2.663850	7.60	9.40	11.100000	13.900000	16.200000
Inflation rate	4424.0	1.228029	1.382711	-0.80	0.30	1.400000	2.600000	3.700000
GDP	4424.0	0.001969	2.269935	-4.06	-1.70	0.320000	1.790000	3.510000

View Data Types

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 35 columns):
 #   Column                                          Non-Null Count  Dtype  
---  ------                                          --------------  -----  
 0   Marital status                                  4424 non-null   int64  
 1   Application mode                                4424 non-null   int64  
 2   Application order                               4424 non-null   int64  
 3   Course                                          4424 non-null   int64  
 4   Daytime/evening attendance                      4424 non-null   int64  
 5   Previous qualification                          4424 non-null   int64  
 6   Nationality                                     4424 non-null   int64  
 7   Mother's qualification                          4424 non-null   int64  
 8   Father's qualification                          4424 non-null   int64  
 9   Mother's occupation                             4424 non-null   int64  
 10  Father's occupation                             4424 non-null   int64  
 11  Displaced                                       4424 non-null   int64  
 12  Educational special needs                       4424 non-null   int64  
 13  Debtor                                          4424 non-null   int64  
 14  Tuition fees up to date                         4424 non-null   int64  
 15  Gender                                          4424 non-null   int64  
 16  Scholarship holder                              4424 non-null   int64  
 17  Age at enrollment                               4424 non-null   int64  
 18  International                                   4424 non-null   int64  
 19  Curricular units 1st sem (credited)             4424 non-null   int64  
 20  Curricular units 1st sem (enrolled)             4424 non-null   int64  
 21  Curricular units 1st sem (evaluations)          4424 non-null   int64  
 22  Curricular units 1st sem (approved)             4424 non-null   int64  
 23  Curricular units 1st sem (grade)                4424 non-null   float64
 24  Curricular units 1st sem (without evaluations)  4424 non-null   int64  
 25  Curricular units 2nd sem (credited)             4424 non-null   int64  
 26  Curricular units 2nd sem (enrolled)             4424 non-null   int64  
 27  Curricular units 2nd sem (evaluations)          4424 non-null   int64  
 28  Curricular units 2nd sem (approved)             4424 non-null   int64  
 29  Curricular units 2nd sem (grade)                4424 non-null   float64
 30  Curricular units 2nd sem (without evaluations)  4424 non-null   int64  
 31  Unemployment rate                               4424 non-null   float64
 32  Inflation rate                                  4424 non-null   float64
 33  GDP                                             4424 non-null   float64
 34  Target                                          4424 non-null   object 
dtypes: float64(5), int64(29), object(1)
memory usage: 1.2+ MB
```

All variables are numeric except for the target variable, which is non-numeric categorical. Will need to convert Target to numeric after exploring the dataset, prior to applying logistic regression.

Missing Data

Use .isna() function to search for any missing data prior to analysis.

In [ ]:
```
print(df.isna().sum())
print('Total Missing: ', df.isna().sum().sum())
```

```
Marital status                                    0
Application mode                                  0
Application order                                 0
Course                                            0
Daytime/evening attendance                        0
Previous qualification                            0
Nationality                                       0
Mother's qualification                            0
Father's qualification                            0
Mother's occupation                               0
Father's occupation                               0
Displaced                                         0
Educational special needs                         0
Debtor                                            0
Tuition fees up to date                           0
Gender                                            0
Scholarship holder                                0
Age at enrollment                                 0
International                                     0
Curricular units 1st sem (credited)               0
Curricular units 1st sem (enrolled)               0
Curricular units 1st sem (evaluations)            0
Curricular units 1st sem (approved)               0
Curricular units 1st sem (grade)                  0
Curricular units 1st sem (without evaluations)    0
Curricular units 2nd sem (credited)               0
Curricular units 2nd sem (enrolled)               0
Curricular units 2nd sem (evaluations)            0
Curricular units 2nd sem (approved)               0
Curricular units 2nd sem (grade)                  0
Curricular units 2nd sem (without evaluations)    0
Unemployment rate                                 0
Inflation rate                                    0
GDP                                               0
Target                                            0
dtype: int64
Total Missing:  0
```

There are no missing values.

Check for Duplicates

In [ ]:
```
print('Total Duplicates: ', df.duplicated().sum())
```

```
Total Duplicates:  0
```

Data Exploration

Check the distribution of the target variable.

In [ ]:
```
df['Target'].value_counts()
```

Out[ ]:
```
Graduate    2209
Dropout     1421
Enrolled     794
Name: Target, dtype: int64
```

To ensure each feature is analyzed in comparison to students who graduate, or drop out only, the 'Enrolled' classification will be dropped from the analysis. Because students who are labelled as "enrolled" still have the opportunity to drop out during their program, whether they drop out or graduate is unknown.

In [ ]:
```
df = df[df.Target != 'Enrolled']
```

Verify updated shape:

In [ ]:
```
df.shape
```

Out[ ]:
```
(3630, 35)
```

After dropping 'Enrolled' values the data frame was reduced from 4424 rows to 3630 rows.

Target Distribution

Next, I will take a look at the current distribution of the target variable with a frequency distribution table and corresponding bar chart.

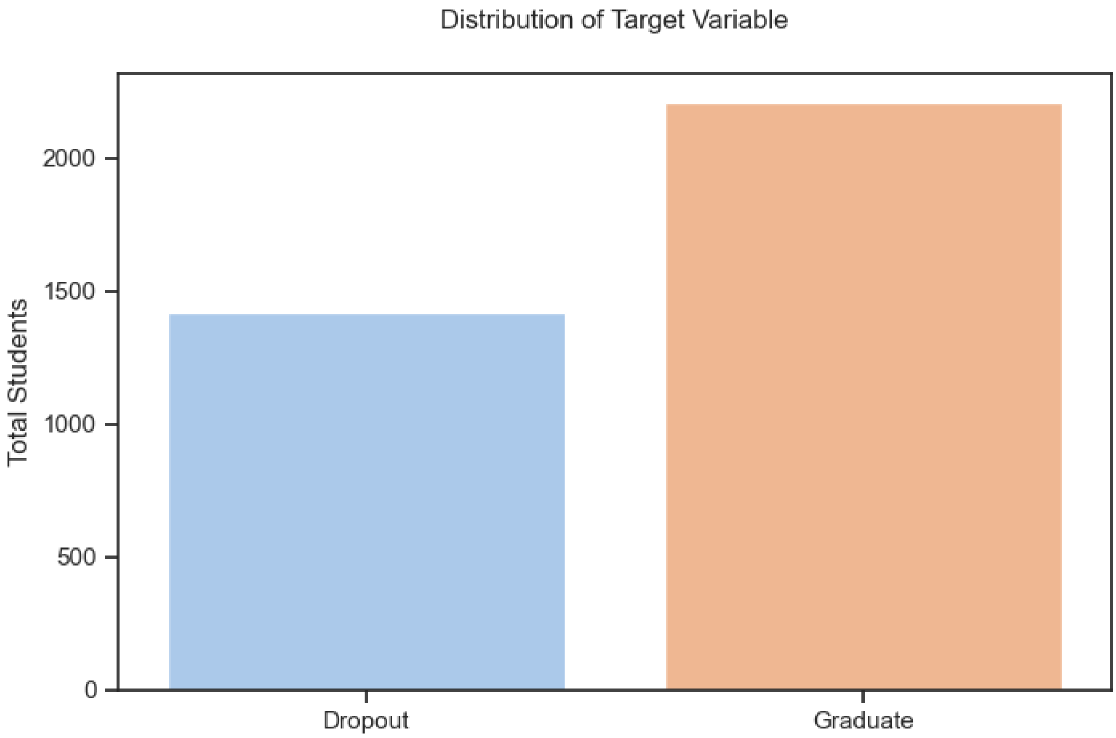
In [ ]:
```
# Create data frame containing % of total column
freq_distribution = pd.DataFrame(df['Target'].value_counts())
freq_distribution['% of Total'] = round((freq_distribution['Target'] / df['Target'].value_counts().sur
freq_distribution
```

Out[ ]:

	Target	% of Total
Graduate	2209	60.85
Dropout	1421	39.15

Approximately 61% of the sample students are labeled as Graduate and 39% are labeled as Dropout.

In [ ]:
```
sns.set_style('ticks')
sns.countplot(df, x='Target', palette='pastel')
plt.ylabel('Total Students')
plt.xlabel(None)
plt.title('Distribution of Target Variable', pad=20)
plt.show()
```



Distribution by Gender

In [ ]:
```
sns.set_style('ticks')
sns.countplot(df, x='Gender', hue='Target', palette='pastel')
plt.xticks(ticks=[0,1], labels=['Female', 'Male'])
plt.ylabel('Total Students')
plt.xlabel(None)
plt.title('Distribution of Target by Gender', pad=20)
plt.show()
```

According to the data there are many more female graduates than male graduates. There appears to be a much larger sample of female students than male students. From this visualization it is easy to see that males are much more likely to drop out than females.

View crosstab of Gender and Target variable:

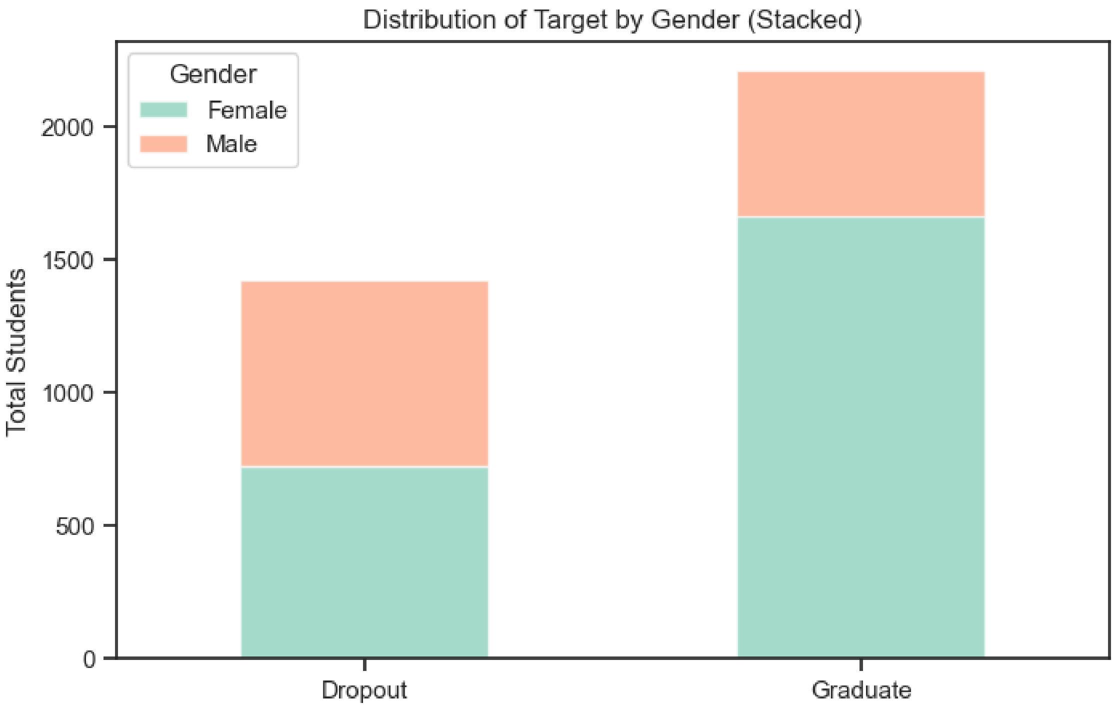
In [ ]:
```
ct = pd.crosstab(df['Target'], df['Gender'])
ct = ct.rename(columns={0: 'Female', 1: 'Male'})
ct
```

Out[ ]:

Gender	**Female**	**Male**
Target		
Dropout	720	701
Graduate	1661	548

Visualize crosstab.

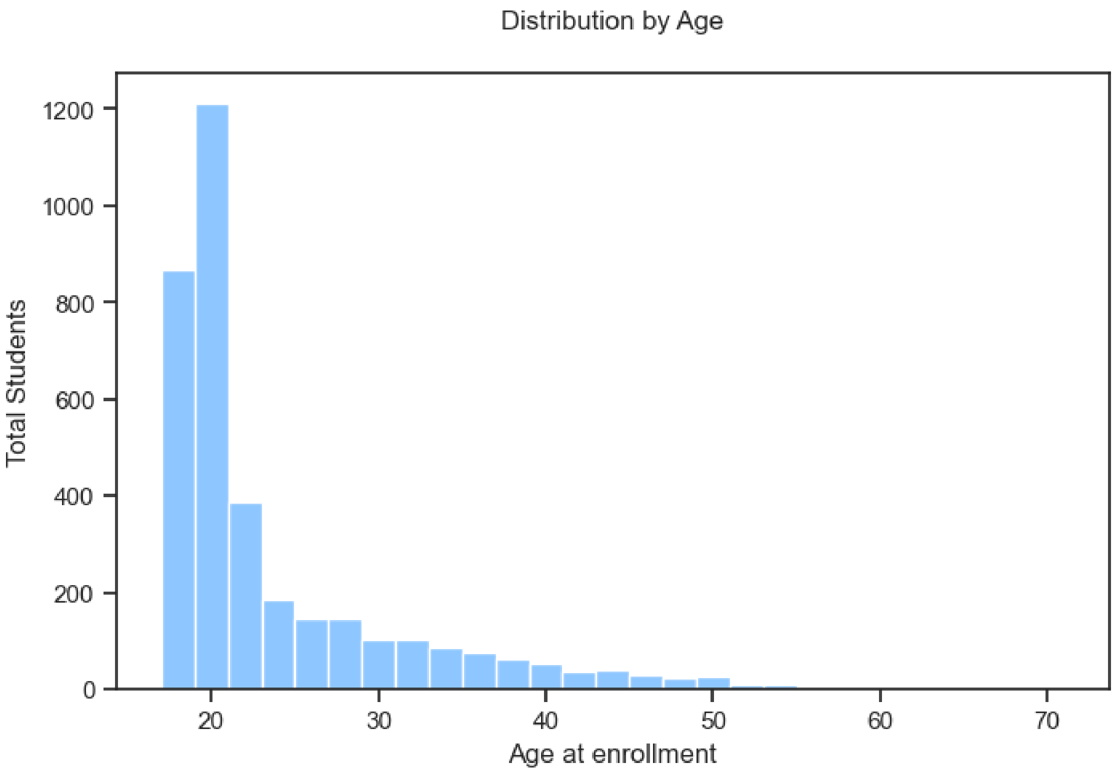
In [ ]:
```
sns.set_style('ticks')
palette = sns.color_palette('Set2')
ct.plot(kind='bar', color=palette, alpha=0.6, stacked=True)

# Customize the labels
plt.xticks(rotation=0)
plt.ylabel('Total Students')
plt.xlabel(None)
plt.title('Distribution of Target by Gender (Stacked)')
plt.show()
```



This visualization makes it easy to see that males account for approximately half of all dropouts from the sample population.

Distribution by Age

In [ ]:
```
sns.set_style('ticks')
sns.histplot(df, x='Age at enrollment', color='dodgerblue', alpha=0.5, binwidth=2)

# Customize the labels
plt.title('Distribution by Age', pad=20)
plt.ylabel('Total Students')

plt.show()
```

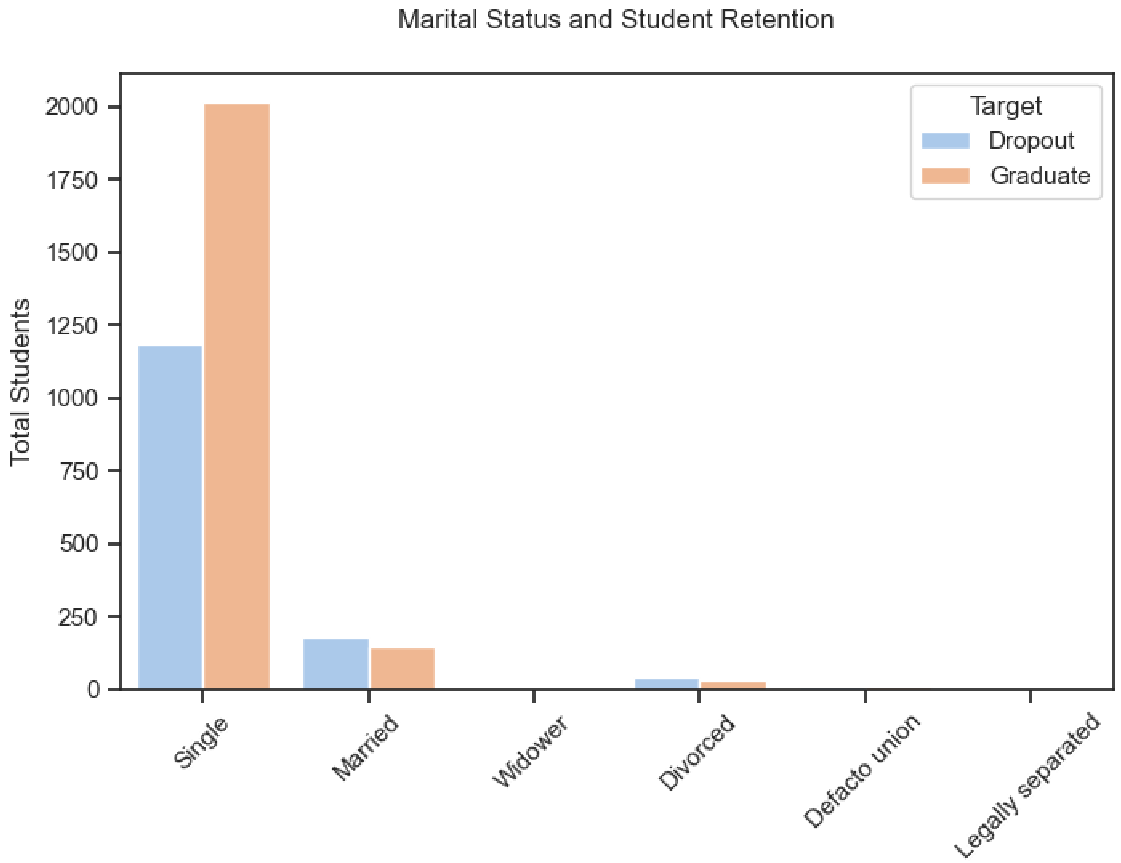


The majority of students in the sample population are between the ages of 18-21, consistent with what we would expect with undergraduate enrollment.

Distribution by Marital Status

In [ ]:
```
sns.set(rc={'figure.figsize':(8, 5)})
sns.set_style('ticks')
sns.countplot(df, x='Marital status', hue='Target', palette='pastel')
```

```
# Change the x tick labels to the corresponding status
plt.xticks(ticks=[0, 1, 2, 3, 4, 5], labels=['Single', 'Married', 'Widower',
                                              'Divorced', 'Defacto union', 'Legally separated'],
                                              rotation=45)
# Customize the labels
plt.ylabel('Total Students')
plt.xlabel(None)
plt.title('Marital Status and Student Retention', pad=20)
plt.show()
```

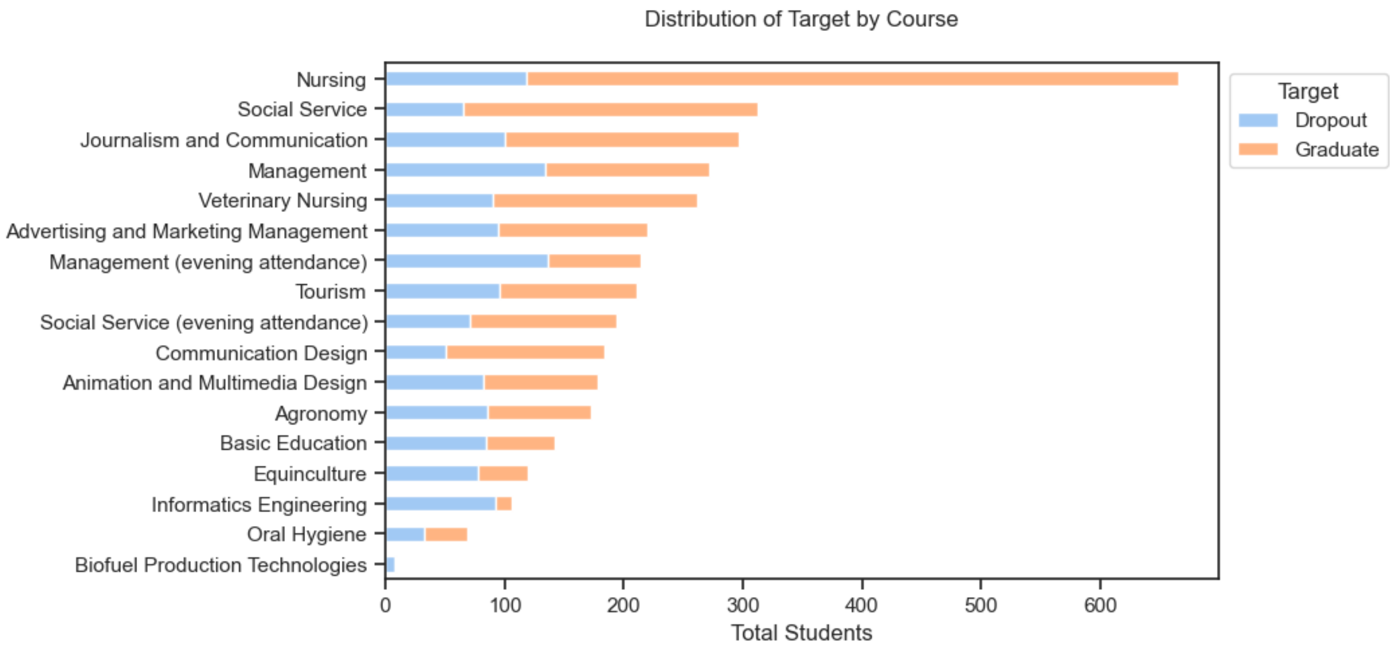


Due to the significant imbalance of marital status leaning toward Single students, marital status is not likely to be a significant influence on overall student success.

Distribution by Course Program

In [ ]:
```
# Group by Course and Target
student_courses = df.groupby(['Course', 'Target']).size().reset_index().pivot(columns='Target', index=
student_courses = student_courses.rename(index={1: 'Biofuel Production Technologies',
                                                2: 'Animation and Multimedia Design',
                                                3: 'Social Service (evening attendance)',
                                                4: 'Agronomy',
                                                5: 'Communication Design',
                                                6: 'Veterinary Nursing',
                                                7: 'Informatics Engineering',
                                                8: 'Equinculture',
                                                9: 'Management',
                                                10: 'Social Service',
                                                11: 'Tourism',
                                                12: 'Nursing',
                                                13: 'Oral Hygiene',
                                                14: 'Advertising and Marketing Management',
                                                15: 'Journalism and Communication',
                                                16: 'Basic Education',
                                                17: 'Management (evening attendance)'})

# Sum the total number of students for each course and sort for the plot
student_courses['Total'] = student_courses.sum(axis=1)
student_courses_sorted = student_courses.sort_values(by='Total', ascending=True)

# Remove the 'Total' column
student_courses_sorted.drop(columns='Total', inplace=True)

# Generate the plot
```

```
sns.set(rc={'figure.figsize':(8, 5)})
sns.set_style('ticks')
sns.set_palette('pastel')
course_plot = student_courses_sorted.plot(kind='barh', stacked=True)

# Customize the labels
plt.title('Distribution of Target by Course', pad=20)
plt.legend(labels=["Dropout", "Graduate"], title='Target', bbox_to_anchor=(1, 1))
plt.xlabel('Total Students')
plt.ylabel(None)
plt.show()

plt.savefig('Student_Course_Distribution.png')
```

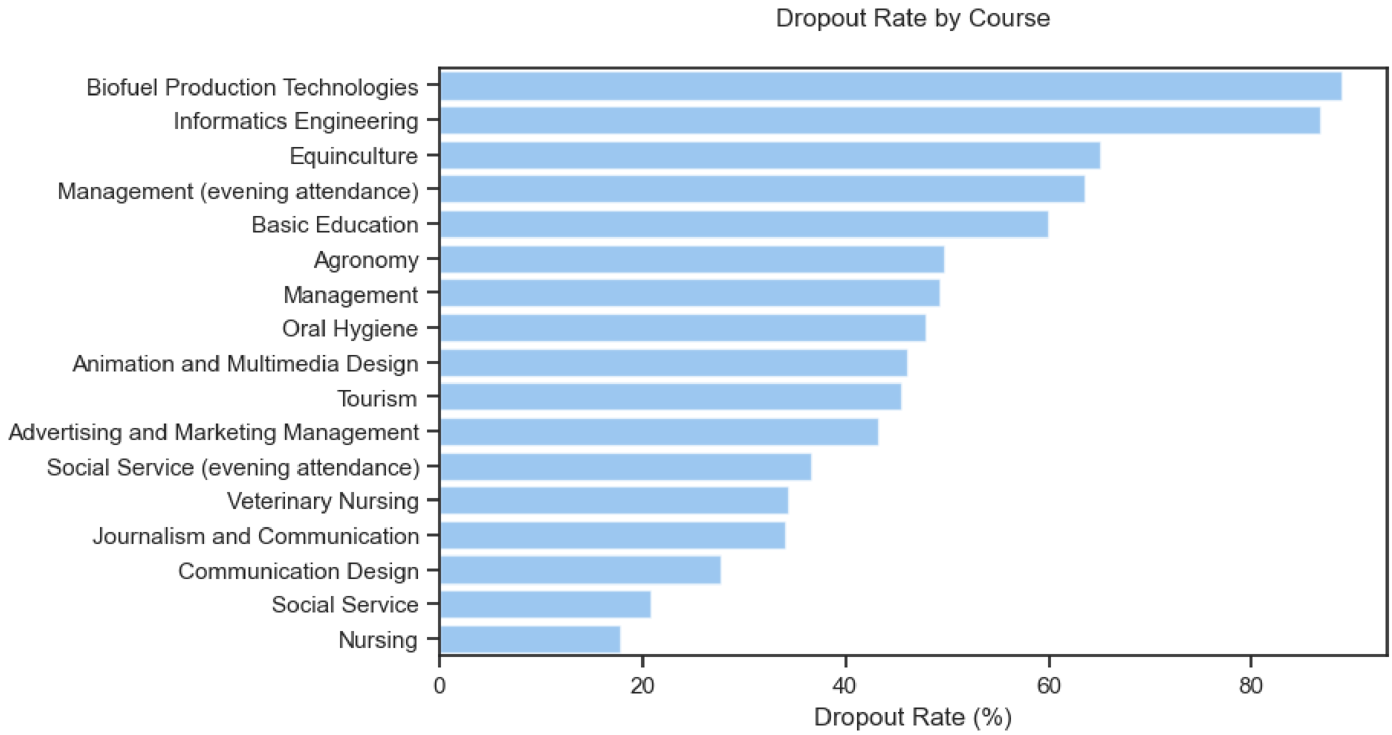



```
<Figure size 800x500 with 0 Axes>
```

Check the actual dropout rates for each course.

In [ ]:
```
course_names = {1: 'Biofuel Production Technologies',
                                                2: 'Animation and Multimedia Design',
                                                3: 'Social Service (evening attendance)',
                                                4: 'Agronomy',
                                                5: 'Communication Design',
                                                6: 'Veterinary Nursing',
                                                7: 'Informatics Engineering',
                                                8: 'Equinculture',
                                                9: 'Management',
                                                10: 'Social Service',
                                                11: 'Tourism',
                                                12: 'Nursing',
                                                13: 'Oral Hygiene',
                                                14: 'Advertising and Marketing Management',
                                                15: 'Journalism and Communication',
                                                16: 'Basic Education',
                                                17: 'Management (evening attendance)'}
df2 = df.copy()
df2['Course'] = df2['Course'].map(course_names)

# Calculate dropout percentages and sort in descending order
dropout_counts = df2.groupby('Course')['Target'].apply(lambda x: (x == 'Dropout').sum())
total_counts = df2['Course'].value_counts()

dropout_percentages = (dropout_counts / total_counts) * 100
dropout_percentages_sorted = dropout_percentages.sort_values(ascending=False)

# Convert to Data Frame
dropout_rate_df = pd.DataFrame({
    'Course': dropout_percentages_sorted.index,
    'Dropout Rate': dropout_percentages_sorted.values
})

# Visualize in a horizontal bar plot
sns.set_style('ticks')
sns.barplot(dropout_rate_df, x='Dropout Rate', y='Course',
```

```
            color='dodgerblue', alpha=0.5)

# Style the plot
plt.title('Dropout Rate by Course', pad=20)
plt.xlabel('Dropout Rate (%)')
plt.ylabel(None)

plt.show()
```



Most courses have more graduates than dropouts, but there are some interesting insights that may indicate some courses are more challenging for students to complete than others. 7 out of 17 courses have over 50% dropout rate.

Feature Selection

Start by transforming target variable into binary numeric by using get_dummies().

In [ ]:
```
df = pd.get_dummies(df, columns=['Target'])
df.head()
```

Out[ ]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nationality	Mother's qualification	Father's qualification	Mother's occupation	Father' occupatio
0	1	8	5	2	1	1	1	13	10	6	1
1	1	6	1	11	1	1	1	1	3	4	
2	1	1	5	5	1	1	1	22	27	10	1
3	1	8	2	15	1	1	1	23	27	6	
4	2	12	1	3	0	1	1	22	28	10	1

Drop the excess dummy variables to retain "Target_Dropout" only and rename back to "Target".

In [ ]:
```
dummies_to_drop = ['Target_Graduate']
df.drop(columns=dummies_to_drop, inplace=True)
df.rename(columns={'Target_Dropout': 'Target'}, inplace=True)

df.head()
```

Out[ ]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nationality	Mother's qualification	Father's qualification	Mother's occupation	Father' occupatio
0	1	8	5	2	1	1	1	13	10	6	1
1	1	6	1	11	1	1	1	1	3	4	
2	1	1	5	5	1	1	1	22	27	10	1
3	1	8	2	15	1	1	1	23	27	6	
4	2	12	1	3	0	1	1	22	28	10	1

Check for collinearity

In [ ]:
```
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

In [ ]:
```
df.corr().round(2)
```

Out[ ]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nationality	Mother's qualification	Father's qualification	Moth occupa
Marital status	1.00	0.24	-0.13	0.00	-0.27	0.13	-0.02	0.18	0.12	(
Application mode	0.24	1.00	-0.25	-0.08	-0.28	0.43	-0.01	0.10	0.09	(
Application order	-0.13	-0.25	1.00	0.12	0.17	-0.20	-0.03	-0.06	-0.05	-(
Course	0.00	-0.08	0.12	1.00	-0.03	-0.16	0.00	0.04	0.03	(
Daytime/evening attendance	-0.27	-0.28	0.17	-0.03	1.00	-0.12	0.03	-0.18	-0.13	-(
Previous qualification	0.13	0.43	-0.20	-0.16	-0.12	1.00	-0.04	0.01	0.02	(
Nationality	-0.02	-0.01	-0.03	0.00	0.03	-0.04	1.00	-0.03	-0.08	(
Mother's qualification	0.18	0.10	-0.06	0.04	-0.18	0.01	-0.03	1.00	0.53	(
Father's qualification	0.12	0.09	-0.05	0.03	-0.13	0.02	-0.08	0.53	1.00	(
Mother's occupation	0.07	0.01	-0.04	0.02	-0.04	0.00	0.02	0.31	0.22	'
Father's occupation	0.03	-0.01	-0.03	0.00	-0.00	0.01	0.02	0.13	0.19	(
Displaced	-0.24	-0.27	0.35	0.01	0.24	-0.17	-0.00	-0.07	-0.07	-(
Educational special needs	-0.03	-0.02	0.03	-0.01	0.03	-0.01	0.00	-0.01	0.00	(
Debtor	0.04	0.11	-0.07	-0.04	0.00	0.12	0.07	0.01	-0.01	(
Tuition fees up to date	-0.10	-0.14	0.06	0.03	0.05	-0.10	-0.05	-0.03	-0.02	-(
Gender	-0.00	0.17	-0.11	-0.09	-0.03	0.11	-0.03	-0.05	-0.06	-(
Scholarship holder	-0.07	-0.16	0.07	0.05	0.11	-0.09	-0.01	0.03	0.09	(
Age at enrollment	0.52	0.47	-0.28	-0.06	-0.45	0.27	-0.01	0.28	0.19	(
International	-0.03	-0.00	-0.03	0.01	0.03	-0.03	0.92	-0.02	-0.07	(
Curricular units 1st sem (credited)	0.07	0.24	-0.13	-0.14	-0.12	0.16	0.01	0.04	0.04	-(
Curricular units 1st sem (enrolled)	0.06	0.16	-0.02	0.11	-0.04	0.08	-0.01	0.05	0.04	(
Curricular units	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nationality	Mother's qualification	Father's qualification	Moth occupa
1st sem (evaluations)	0.06	0.21	-0.09	0.02	-0.05	0.13	-0.00	0.05	0.04	-(
Curricular units 1st sem (approved)	-0.04	-0.03	0.04	0.07	0.03	-0.02	0.00	-0.02	0.01	(
Curricular units 1st sem (grade)	-0.07	-0.12	0.06	0.17	0.07	-0.05	-0.00	-0.04	-0.01	(
Curricular units 1st sem (without evaluations)	0.04	0.05	-0.04	-0.06	0.04	0.04	0.01	0.01	-0.01	-(
Curricular units 2nd sem (credited)	0.07	0.24	-0.13	-0.12	-0.11	0.14	0.00	0.04	0.05	-(
Curricular units 2nd sem (enrolled)	0.04	0.13	0.03	0.18	0.01	0.05	-0.03	0.03	0.03	(
Curricular units 2nd sem (evaluations)	0.03	0.16	-0.04	0.06	0.01	0.08	-0.03	0.03	0.01	-(
Curricular units 2nd sem (approved)	-0.06	-0.08	0.07	0.10	0.05	-0.05	-0.02	-0.02	0.00	(
Curricular units 2nd sem (grade)	-0.08	-0.12	0.06	0.17	0.06	-0.05	-0.01	-0.03	-0.01	(
Curricular units 2nd sem (without evaluations)	0.03	0.05	-0.03	-0.02	-0.01	0.05	-0.01	0.03	0.00	-(
Unemployment rate	-0.02	0.08	-0.10	-0.05	0.07	0.09	-0.00	-0.11	-0.07	(
Inflation rate	0.01	-0.03	-0.00	0.04	-0.02	-0.06	-0.01	0.06	0.06	(
GDP	-0.03	-0.01	0.03	0.01	0.01	0.06	0.03	-0.07	-0.06	(
Target	0.10	0.23	-0.09	-0.01	-0.08	0.10	0.00	0.05	0.00	-(

I see some significant collinearity between some of the academic path variables as well as Nationality/International. I will visualize in a heatmap to see more clearly and decide which variables to omit from the analysis.

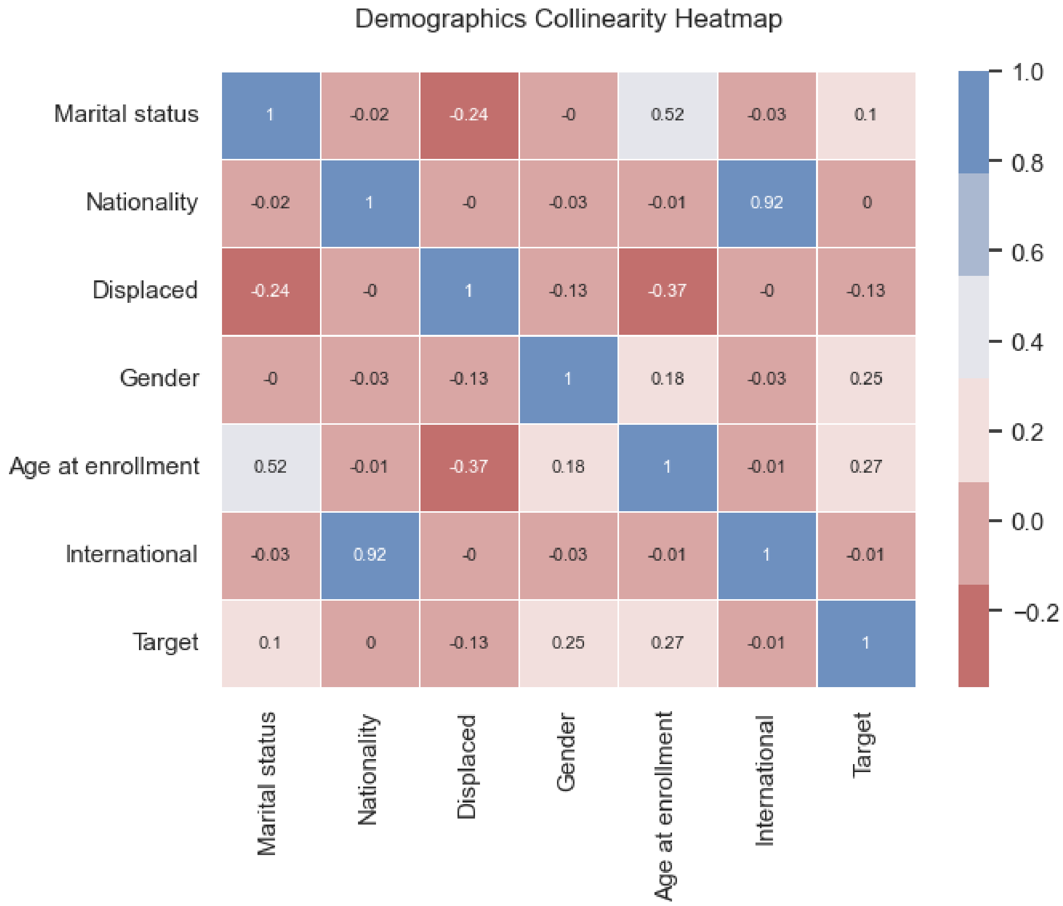
First group the features.

In [ ]:
```
# Demographic
demographics = df[["Marital status", "Nationality", "Displaced", "Gender",
                   "Age at enrollment", "International", "Target"]]
# Academic
academic_path = df[['Curricular units 1st sem (credited)',
       'Curricular units 1st sem (enrolled)',
       'Curricular units 1st sem (evaluations)',
       'Curricular units 1st sem (approved)',
       'Curricular units 1st sem (grade)',
       'Curricular units 1st sem (without evaluations)',
       'Curricular units 2nd sem (credited)',
       'Curricular units 2nd sem (enrolled)',
       'Curricular units 2nd sem (evaluations)',
       'Curricular units 2nd sem (approved)',
       'Curricular units 2nd sem (grade)',
       'Curricular units 2nd sem (without evaluations)',
       'Target']]
```

Demographic Heatmap

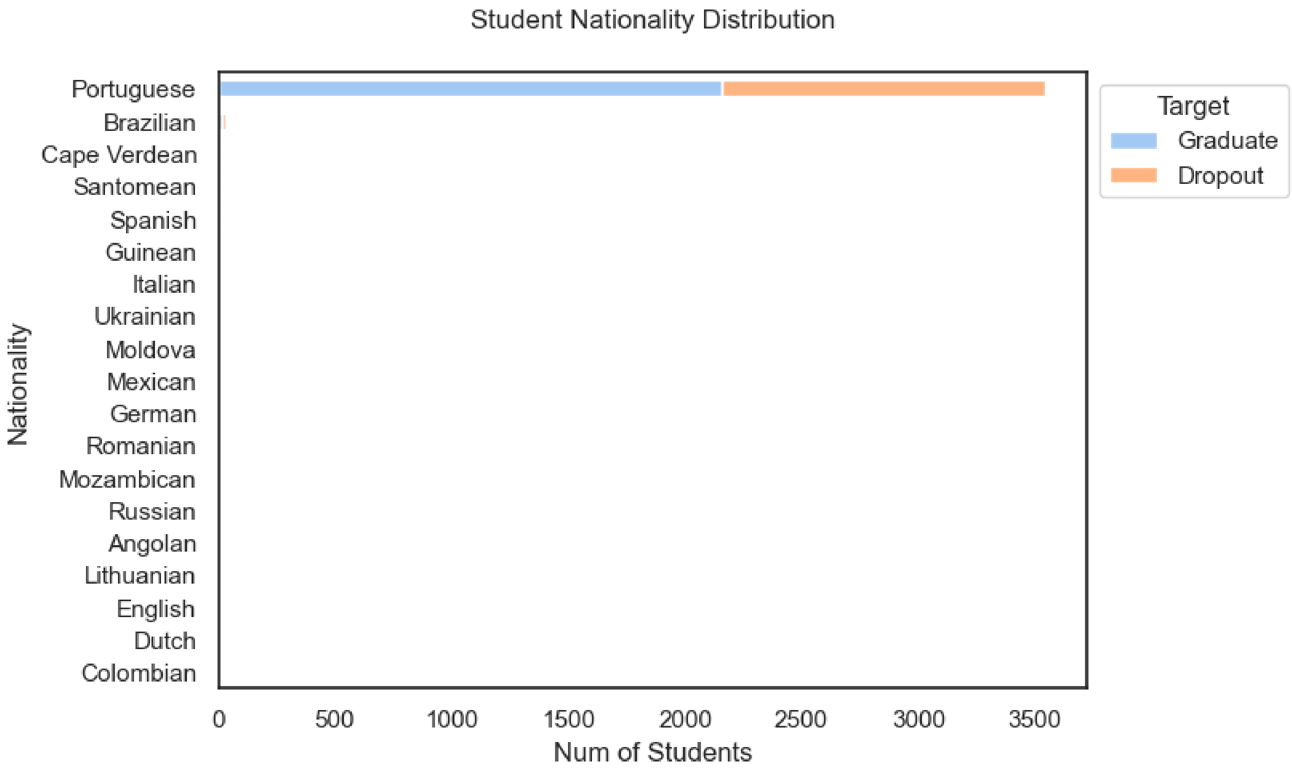
In [ ]:
```
sns.set(rc={"figure.figsize": (7, 5)})
sns.heatmap(demographics.corr().round(2), linewidths=0.5,
            annot=True, annot_kws={"size": 8},
            cmap=sns.color_palette("vlag_r"))

plt.title('Demographics Collinearity Heatmap', pad=20)
```

Out[ ]:Text(0.5, 1.0, 'Demographics Collinearity Heatmap')



Explore Nationality a little further to see what the sample population looks like.

In [ ]:
```
# Group by Nationality and Target
student_nationality = df.groupby(['Nationality', 'Target']).size().reset_index().pivot(columns='Targe
student_nationality = student_nationality.rename(index={1:'Portuguese', 2:'German', 3:'Spanish', 4:'It
                    6:'English', 7:'Lithuanian', 8:'Angolan', 9:'Cape Verdean',
                    10:'Guinean', 11:'Mozambican', 12:'Santomean', 13:'Turkish',
                    14:'Brazilian', 15:'Romanian', 16:'Moldova', 17:'Mexican',
                    18:'Ukrainian', 19:'Russian', 20:'Cuban', 21:'Colombian'})

# Sum the total number of students for each nationality and sort for the plot
student_nationality['Total'] = student_nationality.sum(axis=1)
student_nationality_sorted = student_nationality.sort_values(by='Total', ascending=True)

# Remove the 'Total' column
student_nationality_sorted.drop(columns='Total', inplace=True)

# Generate the plot
sns.set_palette('pastel')
sns.set_style("white")
nationality_plot = student_nationality_sorted.plot(kind='barh', stacked=True)

# Customize the labels
plt.title('Student Nationality Distribution', pad=20)
plt.legend(labels=["Graduate", "Dropout"], title='Target', bbox_to_anchor=(1, 1))
plt.xlabel('Num of Students')
plt.show()

plt.savefig('Student_Nationality_Distribution.png')
```




```
<Figure size 700x500 with 0 Axes>
```

How many students do NOT have Portuguese Nationality?

In [ ]:
```
filtered_student_nationality = student_nationality_sorted[
    (student_nationality_sorted > 0).all(axis=1) &
    (student_nationality_sorted.index != 'Portuguese')
].sort_values(by=0, ascending=False)
filtered_student_nationality
```

Out[ ]:

Target	0	1
Nationality		
Brazilian	18.0	14.0
Santomean	8.0	1.0
Cape Verdean	8.0	4.0
Guinean	4.0	1.0
Spanish	4.0	4.0
Ukrainian	2.0	1.0
Mexican	1.0	1.0

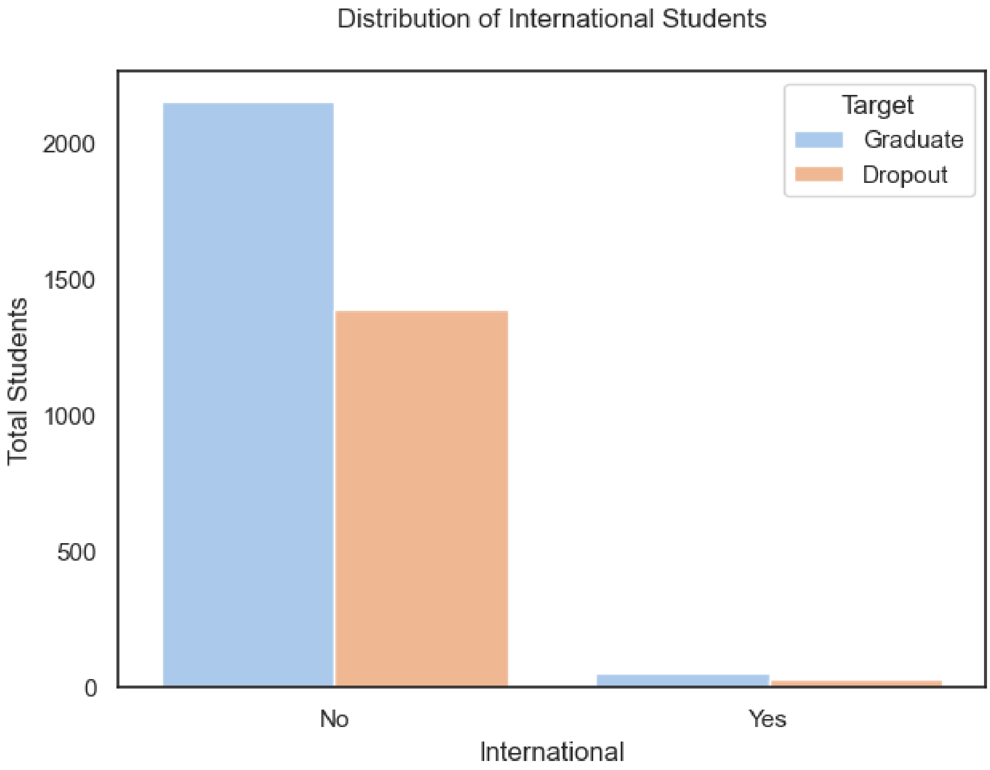
The vast majority of students in this sample population are of Portugese descent, and has 0 correlation with the target variable down to three decimals, which means that this feature will not be a good predictor of student success and can be dropped from the analysis.

Based on the above Nationality data we know the vast majority of the sample population are domestic students, with very few International students.

Display the International distribution plot to validate.

In [ ]:
```
sns.countplot(df, x='International', hue='Target', palette='pastel')

plt.title('Distribution of International Students', pad=20)
plt.ylabel('Total Students')
plt.xticks(ticks=[0,1], labels=['No','Yes'])
plt.legend(labels=["Graduate", "Dropout"], title='Target', bbox_to_anchor=(1, 1))
```

Out[ ]:`<matplotlib.legend.Legend at 0x29091cf10>`



As expected, nearly all students are domestically located, and the International feature has nearly zero correlation with the target variable. Due to this imbalance, the International feature is not a good predictor of student success in this sample population and will be dropped from the analysis.

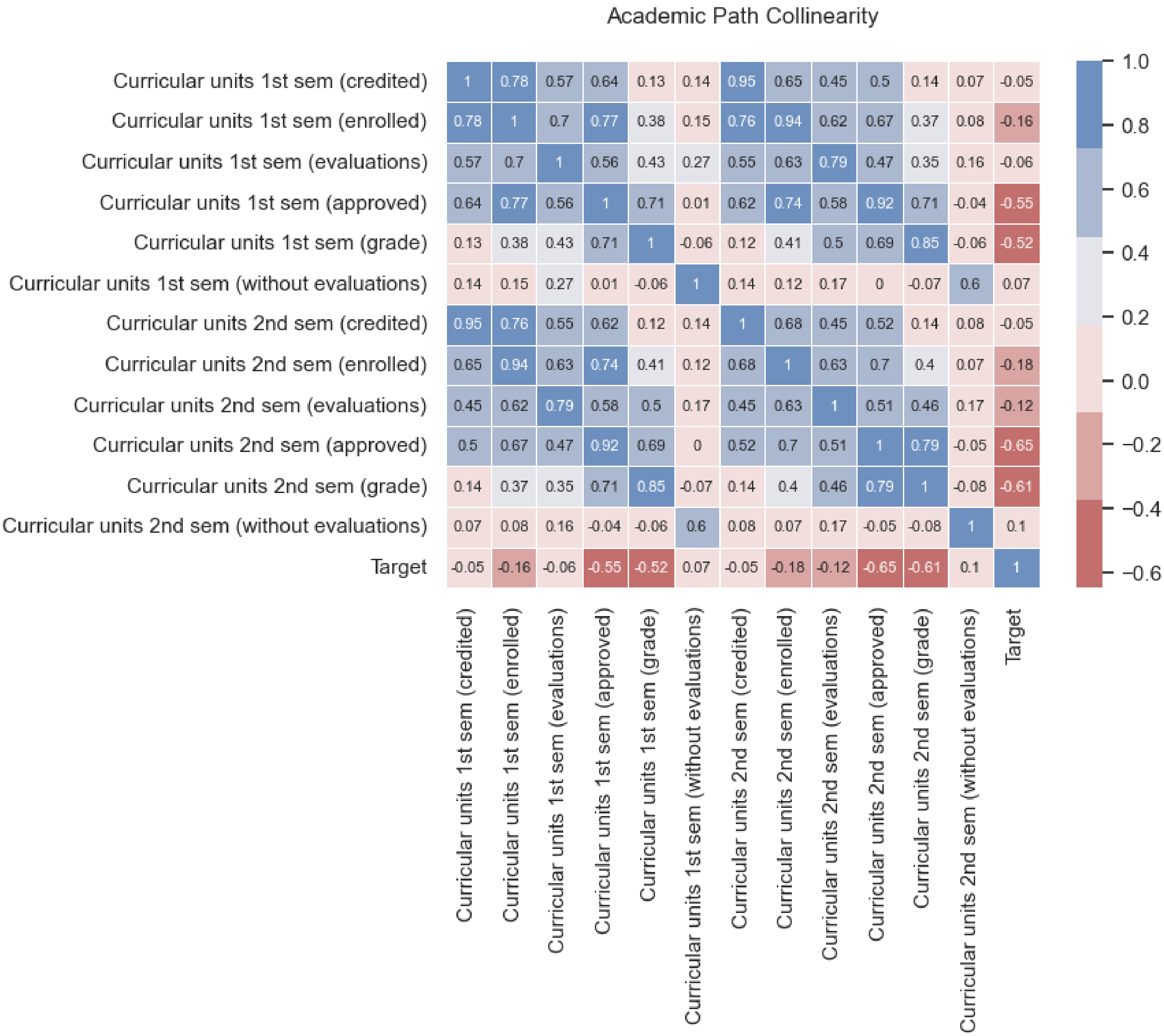
In [ ]:
```
features_to_drop = ['Nationality', 'International']
features_to_drop
```

Out[ ]:`['Nationality', 'International']`

Academic Heatmap

In [ ]:
```
sns.set(rc={"figure.figsize": (7, 5)})
sns.heatmap(academic_path.corr().round(2), linewidths=0.5,
            annot=True, annot_kws={"size": 8},
            cmap=sns.color_palette("vlag_r"))

plt.title('Academic Path Collinearity', pad=20)
```

Out[ ]:Text(0.5, 1.0, 'Academic Path Collinearity')



Some of these academic variables are highly correlated with each other, and some have little to no correlation with the target variable. Based on the above correlation matrix I will drop features with less than 10% correlation with the Target and 1st semester features that are highly correlated with 2nd semester features.

In [ ]:
```
features_to_drop.extend(['Curricular units 1st sem (credited)',
                         'Curricular units 1st sem (enrolled)',
                         'Curricular units 1st sem (evaluations)',
                         'Curricular units 1st sem (approved)',
                         'Curricular units 1st sem (grade)',
                         'Curricular units 1st sem (without evaluations)',
                         'Curricular units 2nd sem (credited)',
                         'Curricular units 2nd sem (without evaluations)']
)
features_to_drop
```

Out[ ]:
```
['Nationality',
 'International',
 'Curricular units 1st sem (credited)',
 'Curricular units 1st sem (enrolled)',
 'Curricular units 1st sem (evaluations)',
 'Curricular units 1st sem (approved)',
 'Curricular units 1st sem (grade)',
 'Curricular units 1st sem (without evaluations)',
 'Curricular units 2nd sem (credited)',
 'Curricular units 2nd sem (without evaluations)']
```

Drop Features

In [ ]:
```
df.drop(features_to_drop, axis=1, inplace=True)
df.head().transpose()
```

Out[ ]:

	0	1	2	3	4
Marital status	1.00	1.000000	1.00	1.00	2.00
Application mode	8.00	6.000000	1.00	8.00	12.00
Application order	5.00	1.000000	5.00	2.00	1.00
Course	2.00	11.000000	5.00	15.00	3.00
Daytime/evening attendance	1.00	1.000000	1.00	1.00	0.00
Previous qualification	1.00	1.000000	1.00	1.00	1.00
Mother's qualification	13.00	1.000000	22.00	23.00	22.00
Father's qualification	10.00	3.000000	27.00	27.00	28.00
Mother's occupation	6.00	4.000000	10.00	6.00	10.00
Father's occupation	10.00	4.000000	10.00	4.00	10.00
Displaced	1.00	1.000000	1.00	1.00	0.00
Educational special needs	0.00	0.000000	0.00	0.00	0.00
Debtor	0.00	0.000000	0.00	0.00	0.00
Tuition fees up to date	1.00	0.000000	0.00	1.00	1.00
Gender	1.00	1.000000	1.00	0.00	0.00
Scholarship holder	0.00	0.000000	0.00	0.00	0.00
Age at enrollment	20.00	19.000000	19.00	20.00	45.00
Curricular units 2nd sem (enrolled)	0.00	6.000000	6.00	6.00	6.00
Curricular units 2nd sem (evaluations)	0.00	6.000000	0.00	10.00	6.00
Curricular units 2nd sem (approved)	0.00	6.000000	0.00	5.00	6.00
Curricular units 2nd sem (grade)	0.00	13.666667	0.00	12.40	13.00
Unemployment rate	10.80	13.900000	10.80	9.40	13.90
Inflation rate	1.40	-0.300000	1.40	-0.80	-0.30
GDP	1.74	0.790000	1.74	-3.12	0.79
Target	1.00	0.000000	1.00	0.00	0.00

Check each remaining variable's correlation with the target variable.

In [ ]:
```
df.corr()['Target']
```

Out[ ]:
```
Marital status                            0.100479
Application mode                          0.233888
Application order                        -0.094355
Course                                   -0.006814
Daytime/evening attendance               -0.084496
Previous qualification                    0.102795
Mother's qualification                    0.048459
Father's qualification                    0.003850
Mother's occupation                      -0.064195
Father's occupation                      -0.073238
Displaced                                -0.126113
Educational special needs                 0.007254
Debtor                                    0.267207
Tuition fees up to date                  -0.442138
Gender                                    0.251955
Scholarship holder                       -0.313018
Age at enrollment                         0.267229
Curricular units 2nd sem (enrolled)      -0.182897
Curricular units 2nd sem (evaluations)   -0.119239
Curricular units 2nd sem (approved)      -0.653995
Curricular units 2nd sem (grade)         -0.605350
Unemployment rate                        -0.004198
Inflation rate                            0.030326
GDP                                      -0.050260
Target                                    1.000000
Name: Target, dtype: float64
```

Since computational time is not an issue, I will leave the remaining features in the analysis.

Logistic Regression

Building The Initial Model

Scale the Data

Access the solution algorithm and instantiate as mm_scaler

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler()
```

Fit the scaler to the predictor variables.

In [ ]:
```
X_scaled = mm_scaler.fit_transform(df)
```

In [ ]:
```
df[df.columns] = X_scaled
df.head()
```

Out[ ]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Mother's qualification	Father's qualification	Mother's occupation	Father's occupation	Displacec
0	0.0	0.411765	0.833333	0.0625	1.0	0.0	0.428571	0.272727	0.161290	0.200000	1.(
1	0.0	0.294118	0.166667	0.6250	1.0	0.0	0.000000	0.060606	0.096774	0.066667	1.(
2	0.0	0.000000	0.833333	0.2500	1.0	0.0	0.750000	0.787879	0.290323	0.200000	1.(
3	0.0	0.411765	0.333333	0.8750	1.0	0.0	0.785714	0.787879	0.161290	0.066667	1.(
4	0.2	0.647059	0.166667	0.1250	0.0	0.0	0.750000	0.818182	0.290323	0.200000	0.(

Create feature and target data structures.

In [ ]:
```
X = df.drop('Target', axis=1)
y = df['Target']
```

Check the structure of X data structure:

In [ ]:
```
X.shape
```

Out[ ]:
```
(3630, 24)
```

Verify structure types:

In [ ]:
```
print("X: ", type(X))
print("y: ", type(y))
```

```
X:  <class 'pandas.core.frame.DataFrame'>
y:  <class 'pandas.core.series.Series'>
```

Fit Model With One Hold-Out Sample

Split the data into train and testing data using sklearn. Save 30% of the data for testing, and stratify the target variable to keep equal proportions in each group.

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30,
                                                    stratify=df['Target'],
                                                    random_state=1)
```

Check that the stratify parameter worked by maintaining the same balance in the train/test data sets as the original data frame:

In [ ]:
```
y_train.value_counts()
```

Out[ ]:
```
0.0    1546
1.0     995
Name: Target, dtype: int64
```

In [ ]:
```
y_test.value_counts()
```

Out[ ]:
```
0.0    663
1.0    426
Name: Target, dtype: int64
```

In [ ]:
```
print("size of X data structures: ", X_train.shape, X_test.shape)
print("size of y data structures: ", y_train.shape, y_test.shape)
```

```
size of X data structures:  (2541, 24) (1089, 24)
size of y data structures:  (2541,) (1089,)
```

In [ ]:
```
print("Proportion of Target in the training data: ", round(995/(995+1546), 3))
print("Proportion of Target in the testing data: ", round(426/(426+663), 3))
```

```
Proportion of Target in the training data:  0.392
Proportion of Target in the testing data:  0.391
```

The data were split as expected.

Access Solution Algorithm

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(solver='lbfgs', max_iter=500)
```

Fit the model to the training data.

In [ ]:
```
logistic_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression(max_iter=500)
```

Show the intercept and coefficients to examine strengths of each feature variable.

In [ ]:
```
print("intercept %.3f" % logistic_model.intercept_, "\n")
cf = pd.DataFrame()
cf['Feature'] = X.columns
cf['Coef'] = np.transpose(logistic_model.coef_).round(3)
cf.sort_values(by='Coef', ascending=False)
```

```
intercept 1.789
```

Out[ ]:

	Feature	Coef
17	Curricular units 2nd sem (enrolled)	4.464
18	Curricular units 2nd sem (evaluations)	3.700
16	Age at enrollment	1.755
3	Course	1.271
12	Debtor	1.127
1	Application mode	1.064
14	Gender	0.467
4	Daytime/evening attendance	0.420
6	Mother's qualification	0.366
10	Displaced	0.151
11	Educational special needs	0.098
23	GDP	0.088
21	Unemployment rate	0.049
2	Application order	-0.002
22	Inflation rate	-0.044
7	Father's qualification	-0.069
5	Previous qualification	-0.168
9	Father's occupation	-0.496
0	Marital status	-0.607
15	Scholarship holder	-0.887
8	Mother's occupation	-0.950
13	Tuition fees up to date	-2.353
20	Curricular units 2nd sem (grade)	-3.206
19	Curricular units 2nd sem (approved)	-9.547

Evaluate Fit

In [ ]:
```
y_fit = logistic_model.predict(X_train)
y_pred = logistic_model.predict(X_test)
```

Check the first few predicted values of y to show a string of forecasted positive and negative values.

In [ ]:
```
print(y_pred[0:25])
```

```
[1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.
 0.]
```

Probabilities for Prediction

Check the probilities of the positive outcomes (Dropout) and combine with the predicted values to quickly assess the model's predictions.

In [ ]:
```
probs = [i[1] for i in logistic_model.predict_proba(X_test)]
pred_df = pd.DataFrame({'true_values': y_test,
'pred_values': y_pred,
'pred_probs':probs})
pred_df.head(15).transpose().style.format("{:.3}")
```

Out[ ]:

	1001	514	1191	383	141	1343	2093	2642	598	323	2548	3168	4045	657	879
true_values	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0
pred_values	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
pred_probs	0.988	0.995	0.118	0.0414	0.142	0.0903	0.0713	0.983	0.163	0.241	0.0491	0.633	0.902	0.0596	0.286

Fit Metrics

Check the accuracy of the model with the train and test data.

In [ ]:
```
from sklearn.metrics import accuracy_score
print('Accuracy for training data: %.3f' % accuracy_score(y_train, y_fit))
print('Accuracy for testing data: %.3f' % accuracy_score(y_test, y_pred))
```

```
Accuracy for training data: 0.901
Accuracy for testing data: 0.907
```

The accuracy is highly similar for both the training and testing data, indicating no overfitting.

Next, display the confusion matrix to see how the model performed in terms of false positives and false negatives compared to true positives and true negatives.

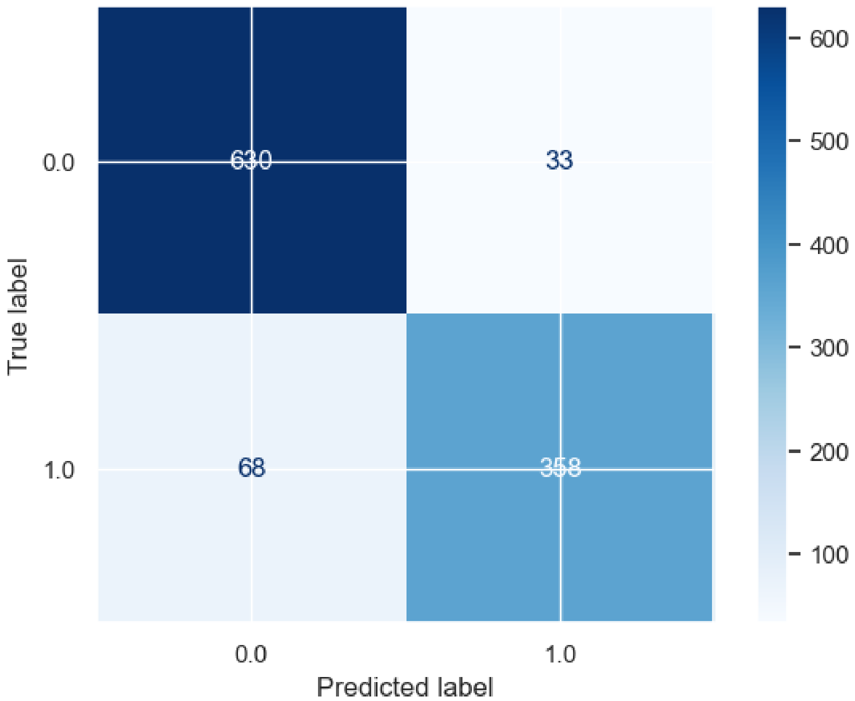
In [ ]:
```
from sklearn.metrics import confusion_matrix
c_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred))
c_matrix
```

Out[ ]:

	0	1
0	630	33
1	68	358

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap='Blues')
```

Out[ ]:`<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28112e770>`



In [ ]:
```
print("True Negatives: ", c_matrix.iloc[0,0])
print("True Positives: ", c_matrix.iloc[1,1])
print("False Negatives: ", c_matrix.iloc[1,0])
print("False Positives: ", c_matrix.iloc[0,1])
```

```
True Negatives:  630
True Positives:  358
False Negatives:  68
False Positives:  33
```

Calculate Recall, Precision, and F1 scores.

In [ ]:
```
from sklearn.metrics import recall_score, precision_score, f1_score
print ('Recall for testing data: %.3f' % recall_score(y_test, y_pred))
print ('Precision for testing data: %.3f' % precision_score(y_test, y_pred))
print ('F1 for testing data: %.3f' % f1_score(y_test, y_pred))
```

```
Recall for testing data: 0.840
Precision for testing data: 0.916
F1 for testing data: 0.876
```

Based on the lowest fit index (recall) 84%, the model correctly forecasts almost 84% of Dropouts as Dropout (true positives). So the model mislabels almost 16% of actual Dropouts as Graduates.

Precision is even higher, which means that of those the model forecasted as Dropout, 91.6% are actual dropouts. 8% of those predicted as Dropout are indicated as Graduate in the data, a false positive.

By definition, the F1 statistic is between recall and precision, their harmonic average, at 87.6%.

Baseline Probabilities

View the baseline probabilities to help evaluate the effectiveness of the model.

In [ ]:
```
my = y.mean()
max_my = np.max([y.mean(), 1-y.mean()])
print("proportion of 0\'s (Graduate): %.3f" % (1-my))
print("Proportion of 1\'s (Dropout): %.3f" % my)
print("Null model accuracy: %.3f" % max_my)
```

```
proportion of 0's (Graduate): 0.609
Proportion of 1's (Dropout): 0.391
Null model accuracy: 0.609
```

The Logistic regression model with a single hold-out sample is significantly more accurate than the null model.

Model Validation

Next I will evaluate the effectiveness of the model with multiple hold-out samples using K-fold cross-validation.

In [ ]:
```
from sklearn.model_selection import StratifiedKFold # access solution algorithm and instantiate
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

In [ ]:
```
logistic_model = LogisticRegression(solver='lbfgs', max_iter=1000)
```

In [ ]:
```
from sklearn.model_selection import cross_validate
scores = cross_validate(logistic_model, X, y, cv=skf,
                        scoring=('accuracy', 'recall', 'precision'),
                        return_train_score=True)
```

Display the scores as a data frame.

In [ ]:
```
logistic_scores = pd.DataFrame(scores).round(3)
logistic_scores
```

Out[ ]:

	fit_time	score_time	test_accuracy	train_accuracy	test_recall	train_recall	test_precision	train_precision
0	0.034	0.005	0.901	0.906	0.807	0.827	0.931	0.925
1	0.016	0.002	0.904	0.905	0.849	0.823	0.899	0.925
2	0.025	0.002	0.906	0.904	0.831	0.821	0.922	0.927
3	0.020	0.004	0.893	0.905	0.806	0.826	0.909	0.923
4	0.024	0.002	0.894	0.901	0.799	0.824	0.919	0.914

Display the mean values for the cross-validation fit metrics.

In [ ]:
```
print('Mean of test accuracy: %.3f' % logistic_scores['test_accuracy'].mean())
print('Mean of test recall: %.3f' % logistic_scores['test_recall'].mean())
print('Mean of test precision: %.3f' % logistic_scores['test_precision'].mean())
```

```
Mean of test accuracy: 0.900
Mean of test recall: 0.818
Mean of test precision: 0.916
```

The result of the cross-validation with multiple hold-out samples is a slight decrease accuracy and recall, but overall no significant change in the conclusion of a good-fitting model.

Original fit scores:
Accuracy for testing data: 0.907
Recall for testing data: 0.840
Precision for testing data: 0.916

Decision Tree

Next, I will test how effective a decision tree model is at predicting the target variable compared to logistic regression. From this information I will be able to determine the best model for predicting student success.

Reestablish X and y data structures and class labels for the final tree.

In [ ]:
```
classes = ['Graduate', 'Dropout']
X = df.drop(['Target'], axis=1)
y = df['Target']
```

In [ ]:
```
X.head().transpose()
```

Out[ ]:

	0	1	2	3	4
Marital status	0.000000	0.000000	0.000000	0.000000	0.200000
Application mode	0.411765	0.294118	0.000000	0.411765	0.647059
Application order	0.833333	0.166667	0.833333	0.333333	0.166667
Course	0.062500	0.625000	0.250000	0.875000	0.125000
Daytime/evening attendance	1.000000	1.000000	1.000000	1.000000	0.000000
Previous qualification	0.000000	0.000000	0.000000	0.000000	0.000000
Mother's qualification	0.428571	0.000000	0.750000	0.785714	0.750000
Father's qualification	0.272727	0.060606	0.787879	0.787879	0.818182
Mother's occupation	0.161290	0.096774	0.290323	0.161290	0.290323
Father's occupation	0.200000	0.066667	0.200000	0.066667	0.200000
Displaced	1.000000	1.000000	1.000000	1.000000	0.000000
Educational special needs	0.000000	0.000000	0.000000	0.000000	0.000000
Debtor	0.000000	0.000000	0.000000	0.000000	0.000000
Tuition fees up to date	1.000000	0.000000	0.000000	1.000000	1.000000
Gender	1.000000	1.000000	1.000000	0.000000	0.000000
Scholarship holder	0.000000	0.000000	0.000000	0.000000	0.000000
Age at enrollment	0.056604	0.037736	0.037736	0.056604	0.528302
Curricular units 2nd sem (enrolled)	0.000000	0.260870	0.260870	0.260870	0.260870
Curricular units 2nd sem (evaluations)	0.000000	0.181818	0.000000	0.303030	0.181818
Curricular units 2nd sem (approved)	0.000000	0.300000	0.000000	0.250000	0.300000
Curricular units 2nd sem (grade)	0.000000	0.735897	0.000000	0.667692	0.700000
Unemployment rate	0.372093	0.732558	0.372093	0.209302	0.732558
Inflation rate	0.488889	0.111111	0.488889	0.000000	0.111111
GDP	0.766182	0.640687	0.766182	0.124174	0.640687

Access Solution Algorithm

Accesss the DecisionTreeClassifier and instantiate with a max_depth of 5.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=5)
dt_model
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

Grid Search: Hyperparameter Tuning with Cross-Validation

Access Kfold module with 3 data splits.

In [ ]:
```
from sklearn.model_selection import KFold
kf3 = KFold(n_splits=3, shuffle=True, random_state=1)
```

Import GridSearchCV and set the parameters.

In [ ]:
```
from sklearn.model_selection import GridSearchCV
kparams = {'max_depth': [2, 3, 4, 5],
           'max_features': [2, 3, 4, 5, 6, 7, 8]}
grid_search = GridSearchCV(dt_model, param_grid=kparams, cv=kf3,
                           scoring=('accuracy', 'recall', 'precision'),
                           refit=False, return_train_score=True)
```

```
grid_search.fit(X,y)
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [ ]:
```
dt_results = pd.DataFrame(grid_search.cv_results_).round(3)
dt_results = dt_results.drop(['params'], axis='columns')
```

In [ ]:
```
# Ensure all rows are displayed
pd.set_option('display.max_rows', None)
```

In [ ]:
```
# Select relevant columns
dt_summary = dt_results[['param_max_depth', 'param_max_features',
'mean_test_accuracy', 'mean_test_recall', 'mean_test_precision',
'mean_train_accuracy','mean_train_recall', 'mean_train_precision']]

# Shorten column names for readability
dt_summary = dt_summary.rename(columns= {
'param_max_depth': 'depth',
'param_max_features': 'features',
'mean_test_accuracy': 'test_accuracy',
'mean_test_recall': 'test_recall',
'mean_test_precision': 'test_precision',
'mean_train_accuracy': 'train_accuracy',
'mean_train_recall': 'train_recall',
'mean_train_precision': 'train_precision'})

# Add columns to quickly assess varance in train vs. test metrics
dt_summary['accuracy_variance'] = abs(dt_summary['train_accuracy'] - dt_summary['test_accuracy'])
dt_summary['recall_variance'] = abs(dt_summary['train_recall'] - dt_summary['test_recall'])
dt_summary['precision_variance'] = abs(dt_summary['train_precision'] - dt_summary['test_precision'])
dt_summary
```

Out[ ]:

	depth	features	test_accuracy	test_recall	test_precision	train_accuracy	train_recall	train_precision	accuracy_variance	recall_varia
0	2	2	0.672	0.451	0.624	0.679	0.459	0.637	0.007	0.
1	2	3	0.733	0.563	0.690	0.735	0.557	0.704	0.002	0.
2	2	4	0.722	0.506	0.746	0.724	0.496	0.758	0.002	0.
3	2	5	0.751	0.498	0.832	0.759	0.528	0.821	0.008	0.
4	2	6	0.775	0.482	0.893	0.773	0.479	0.898	0.002	0.
5	2	7	0.786	0.491	0.927	0.787	0.490	0.942	0.001	0.
6	2	8	0.852	0.765	0.846	0.857	0.774	0.850	0.005	0.
7	3	2	0.765	0.526	0.792	0.775	0.550	0.804	0.010	0.
8	3	3	0.875	0.740	0.926	0.881	0.753	0.930	0.006	0.
9	3	4	0.845	0.679	0.900	0.844	0.678	0.900	0.001	0.
10	3	5	0.828	0.671	0.858	0.826	0.655	0.865	0.002	0.
11	3	6	0.847	0.713	0.879	0.861	0.748	0.883	0.014	0.
12	3	7	0.880	0.794	0.888	0.880	0.789	0.894	0.000	0.
13	3	8	0.849	0.694	0.894	0.853	0.699	0.899	0.004	0.
14	4	2	0.806	0.635	0.831	0.809	0.633	0.841	0.003	0.
15	4	3	0.803	0.587	0.861	0.814	0.608	0.875	0.011	0.
16	4	4	0.831	0.645	0.904	0.840	0.663	0.913	0.009	0.
17	4	5	0.874	0.770	0.894	0.881	0.781	0.904	0.007	0.
18	4	6	0.877	0.780	0.895	0.886	0.799	0.900	0.009	0.
19	4	7	0.869	0.742	0.907	0.871	0.747	0.908	0.002	0.
20	4	8	0.884	0.787	0.905	0.888	0.786	0.914	0.004	0.
21	5	2	0.840	0.693	0.871	0.855	0.712	0.894	0.015	0.
22	5	3	0.823	0.667	0.845	0.840	0.692	0.873	0.017	0.
23	5	4	0.875	0.777	0.892	0.894	0.801	0.920	0.019	0.
24	5	5	0.854	0.723	0.884	0.876	0.763	0.907	0.022	0.
25	5	6	0.885	0.794	0.901	0.893	0.799	0.917	0.008	0.
26	5	7	0.861	0.738	0.888	0.881	0.762	0.922	0.020	0.
27	5	8	0.883	0.775	0.912	0.896	0.786	0.938	0.013	0.

Based on the grid search above, a decision tree model with a depth of 4 and 8 features seems to fit the data reasonably well, with the highest recall and minimal variance between the train/test data. I will use this model to test on the data.

In [ ]:
```
dt_model = DecisionTreeClassifier(max_depth=4, max_features=8)
dt_fit = dt_model.fit(X,y) # Save the results of the .fit() for use in the Tree later.
```

Check Feature Importance

In [ ]:
```
dImp = pd.DataFrame(dt_model.feature_importances_)
dImp = dImp.set_index(X.columns, drop=False)
dImp.columns = ['Importance']
dImp.round(3)
```

Out[ ]:

	Importance
Marital status	0.000
Application mode	0.000
Application order	0.000
Course	0.000
Daytime/evening attendance	0.000
Previous qualification	0.000
Mother's qualification	0.000
Father's qualification	0.000
Mother's occupation	0.001
Father's occupation	0.000
Displaced	0.000
Educational special needs	0.000
Debtor	0.002
Tuition fees up to date	0.296
Gender	0.000
Scholarship holder	0.004
Age at enrollment	0.000
Curricular units 2nd sem (enrolled)	0.049
Curricular units 2nd sem (evaluations)	0.013
Curricular units 2nd sem (approved)	0.623
Curricular units 2nd sem (grade)	0.008
Unemployment rate	0.000
Inflation rate	0.000
GDP	0.003

Age at enrollment and Curricular units 2nd sem(approved) hold the most weight in the model.

Compute predicted values with the final model.

In [ ]:
```
y_fit = dt_model.predict(X)
y_fit[0:19]
```

Out[ ]:
```
array([0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0.,
       0., 0.])
```

View confusion matrix.

In [ ]:
```
pd.DataFrame(confusion_matrix(y, y_fit))
```

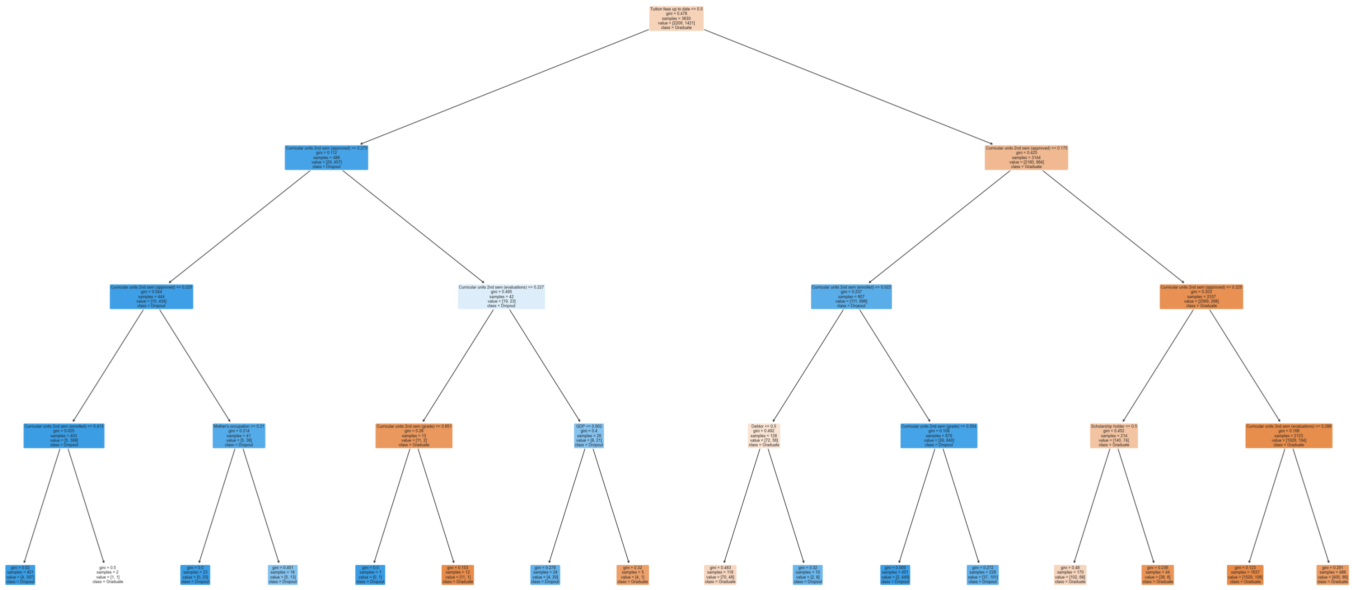
Out[ ]:

	0	1
0	2155	54
1	319	1102

Visualize The Tree

In [ ]:
```
from sklearn import tree
```

In [ ]:
```
plt.figure(figsize=(40,20))
tree.plot_tree(dt_fit, feature_names = X.columns.tolist(),
               class_names=['Graduate', 'Dropout'],
               rounded=True, filled=True)
plt.savefig('dt_Dropout.png')
```



Visualize as text to make it easier to read.

In [ ]:
```
# access export_text module from sklearn
from sklearn.tree import export_text
```

In [ ]:
```
text = export_text(dt_fit, feature_names=X.columns.tolist())
print(text)
```

```
|--- Tuition fees up to date <= 0.50
|   |--- Curricular units 2nd sem (approved) <= 0.28
|   |   |--- Curricular units 2nd sem (approved) <= 0.23
|   |   |   |--- Curricular units 2nd sem (enrolled) <= 0.41
|   |   |   |   |--- class: 1.0
|   |   |   |--- Curricular units 2nd sem (enrolled) >  0.41
|   |   |   |   |--- class: 0.0
|   |   |--- Curricular units 2nd sem (approved) >  0.23
|   |   |   |--- Mother's occupation <= 0.21
|   |   |   |   |--- class: 1.0
|   |   |   |--- Mother's occupation >  0.21
|   |   |   |   |--- class: 1.0
|   |--- Curricular units 2nd sem (approved) >  0.28
|   |   |--- Curricular units 2nd sem (evaluations) <= 0.23
|   |   |   |--- Curricular units 2nd sem (grade) <= 0.65
|   |   |   |   |--- class: 1.0
|   |   |   |--- Curricular units 2nd sem (grade) >  0.65
|   |   |   |   |--- class: 0.0
|   |   |--- Curricular units 2nd sem (evaluations) >  0.23
|   |   |   |--- GDP <= 0.90
|   |   |   |   |--- class: 1.0
|   |   |   |--- GDP >  0.90
|   |   |   |   |--- class: 0.0
|--- Tuition fees up to date >  0.50
|   |--- Curricular units 2nd sem (approved) <= 0.18
|   |   |--- Curricular units 2nd sem (enrolled) <= 0.02
|   |   |   |--- Debtor <= 0.50
|   |   |   |   |--- class: 0.0
|   |   |   |--- Debtor >  0.50
|   |   |   |   |--- class: 1.0
|   |   |--- Curricular units 2nd sem (enrolled) >  0.02
|   |   |   |--- Curricular units 2nd sem (grade) <= 0.55
|   |   |   |   |--- class: 1.0
|   |   |   |--- Curricular units 2nd sem (grade) >  0.55
|   |   |   |   |--- class: 1.0
|   |--- Curricular units 2nd sem (approved) >  0.18
|   |   |--- Curricular units 2nd sem (approved) <= 0.23
|   |   |   |--- Scholarship holder <= 0.50
|   |   |   |   |--- class: 0.0
|   |   |   |--- Scholarship holder >  0.50
|   |   |   |   |--- class: 0.0
|   |   |--- Curricular units 2nd sem (approved) >  0.23
|   |   |   |--- Curricular units 2nd sem (evaluations) <= 0.29
|   |   |   |   |--- class: 0.0
|   |   |   |--- Curricular units 2nd sem (evaluations) >  0.29
|   |   |   |   |--- class: 0.0
```

Decision Tree Accuracy: 0.884
Decision Tree Recall: 0.787
Decision Tree Precision: 0.905

Logistic Accuracy: 0.906
Logistic Recall: 0.834
Logistic Precision: 0.918

With all of the fit indicies (accuracy, recall, & precision) lower than the logistic regression model, this analysis indicates that the logistic regression model performs better than the decision tree model.

Random Forest Analysis

One last model I would like to try is the random forest to see if I can improve upon the results of the decision tree model. The random forest will help by reducing the randomness of the decision tree and provide reduced sensitivity to noise in the data.

Access Solution Algorithm

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(max_depth=5)
rf_model
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5)
```

Build the Model

In [ ]:
```
kf3 = KFold(n_splits=3, shuffle=True, random_state=1)
```

```python
params = {'max_depth': [2, 3, 4, 5],
'max_features': [1, 2, 3, 4, 5, 6]}
grid_search = GridSearchCV(rf_model, param_grid=params, cv=kf3,
scoring=('accuracy', 'recall', 'precision'),
refit=False, return_train_score=True)
grid_search.fit(X,y)
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
▸ RandomForestClassifier
```

Display the results.

In [ ]:

```python
# Adds results to a data frame
rf_results = pd.DataFrame(grid_search.cv_results_).round(3)
rf_results = rf_results.drop(['params'], axis='columns')
```

In [ ]:

```python
# Select relevant columns
rf_summary = rf_results[['param_max_depth', 'param_max_features', 'mean_test_accuracy',
'mean_test_recall', 'mean_test_precision', 'mean_train_accuracy',
'mean_train_recall', 'mean_train_precision']]

# Shorten column names for readability
rf_summary = rf_summary.rename(columns= {
'param_max_depth': 'depth',
'param_max_features': 'features',
'mean_test_accuracy': 'test_accuracy',
'mean_test_recall': 'test_recall',
'mean_test_precision': 'test_precision',
'mean_train_accuracy': 'train_accuracy',
'mean_train_recall': 'train_recall',
'mean_train_precision': 'train_precision'})

# Add columns to quickly assess varance in train vs. test metrics
rf_summary['accuracy_variance'] = abs(rf_summary['train_accuracy'] - rf_summary['test_accuracy'])
rf_summary['recall_variance'] = abs(rf_summary['train_recall'] - rf_summary['test_recall'])
rf_summary['precision_variance'] = abs(rf_summary['train_precision'] - rf_summary['test_precision'])
rf_summary
```

Out[ ]:

	depth	features	test_accuracy	test_recall	test_precision	train_accuracy	train_recall	train_precision	accuracy_variance	recall_varia
0	2	1	0.753	0.378	0.975	0.757	0.389	0.977	0.004	0.
1	2	2	0.842	0.643	0.934	0.842	0.640	0.937	0.000	0.
2	2	3	0.862	0.706	0.924	0.867	0.720	0.923	0.005	0.
3	2	4	0.870	0.735	0.917	0.871	0.738	0.918	0.001	0.
4	2	5	0.875	0.756	0.910	0.876	0.754	0.914	0.001	0.
5	2	6	0.877	0.764	0.908	0.877	0.766	0.905	0.000	0.
6	3	1	0.817	0.570	0.937	0.826	0.590	0.947	0.009	0.
7	3	2	0.870	0.732	0.919	0.875	0.741	0.926	0.005	0.
8	3	3	0.877	0.755	0.917	0.882	0.764	0.922	0.005	0.
9	3	4	0.883	0.779	0.909	0.887	0.786	0.913	0.004	0.
10	3	5	0.884	0.784	0.906	0.889	0.796	0.910	0.005	0.
11	3	6	0.888	0.803	0.901	0.892	0.809	0.906	0.004	0.
12	4	1	0.844	0.638	0.946	0.849	0.644	0.956	0.005	0.
13	4	2	0.882	0.765	0.921	0.887	0.774	0.926	0.005	0.
14	4	3	0.889	0.787	0.917	0.894	0.793	0.924	0.005	0.
15	4	4	0.892	0.796	0.918	0.899	0.805	0.928	0.007	0.
16	4	5	0.891	0.801	0.910	0.899	0.813	0.920	0.008	0.
17	4	6	0.895	0.806	0.917	0.901	0.813	0.924	0.006	0.
18	5	1	0.864	0.708	0.926	0.880	0.739	0.941	0.016	0.
19	5	2	0.884	0.764	0.928	0.899	0.788	0.944	0.015	0.
20	5	3	0.894	0.795	0.922	0.905	0.810	0.937	0.011	0.
21	5	4	0.897	0.809	0.919	0.911	0.822	0.943	0.014	0.
22	5	5	0.898	0.809	0.921	0.909	0.822	0.937	0.011	0.
23	5	6	0.898	0.805	0.924	0.909	0.818	0.942	0.011	0.

Choose the model with a depth of 3 and 6 features, as it is the simplest model with high fit metrics and little variance between the train and test data.

Random Forest Accuracy: 0.888
Random Forest Recall: 0.803
Random Forest Precision: 0.901

Decision Tree Accuracy: 0.884
Decision Tree Recall: 0.787
Decision Tree Precision: 0.905

Logistic Accuracy: 0.906
Logistic Recall: 0.834
Logistic Precision: 0.918